珠峰前端架构课

想参加珠峰架构课正式同学可以咨询右下角客服老师微信：正式课上课时间是周三周五晚8-10点，周日全天。珠峰专注前端十二年

一.打包完整组件库

> 在components下创建入口文件导出所有的组件

```
export * from './icon'
```

> 创建打包组件库的入口

```
mkdir z-plus && cd z-plus
pnpm init
```

```
import { App } from "vue";
import ZIcon from "@z-plus/components/icon";
const components = [ZIcon];
const install = (app: App) => {
  components.forEach((component) => app.use(component));
};
export default {
  install,
};
export * from "@z-plus/components";
```

> 打包组件库

```
export default series(
    withTaskName('clean',()=> run('rm -rf ./dist')),
    parallel(
        withTaskName('buildFullComponent',()=> run('pnpm run build
buildFullComponent')),
    )
)
 export * from './full-component'
```

> 安装打包所需依赖

```
pnpm install rollup @rollup/plugin-node-resolve @rollup/plugin-commonjs rollup-plugin-typescript2 rollup-plugin-vue -D -w
```

> 打包`umd`和`es`模块

```
export const outDir = path.resolve(projectRoot, "dist");
export const zpRoot = path.resolve(projectRoot, "packages/z-plus");
```

`build/full-component.ts`

```
import { parallel } from "gulp";
import path from "path";
import { rollup, OutputOptions } from "rollup";
import { nodeResolve } from "@rollup/plugin-node-resolve";
import commonjs from "@rollup/plugin-commonjs";
import typescript from "rollup-plugin-typescript2";
import vue from "rollup-plugin-vue";
import { projectRoot, zpRoot } from "./utils/paths";
const buildFull = async () => {
  const config = {
    input: path.resolve(zpRoot, 'index.ts'),
    plugins: [nodeResolve(), typescript(), vue(), commonjs()],
    external(id) {
      // 排除vue本身
      return /^vue/.test(id);
    },
  };
  const bundle = await rollup(config);
  const buildConfig = [
    {
      format: "umd",
      file: path.resolve(projectRoot, "dist/index.js"),
      name: "ZPlus",
      exports: "named",
      globals: {
        vue: "Vue",
      },
    },
    {
      format: "esm",
      file: path.resolve(projectRoot, "dist/index.esm.js"),
    },
  ];
  return Promise.all(
    buildConfig.map((config) => bundle.write(config as OutputOptions))
  );
};
export const buildFullComponent = parallel(buildFull);
```

二.对组件依次打包

```
pnpm install fast-glob -w -D
```

```
withTaskName("buildComponent", () => run("pnpm run build buildComponent"));
```

> 找到components下所有的组件对组件依次进行打包

```
export const compRoot = path.resolve(projectRoot, "packages/components");
```

> `build/component.ts`

```
import { series } from "gulp";
import path from "path";
import { sync } from "fast-glob";
import { nodeResolve } from "@rollup/plugin-node-resolve";
import vue from "rollup-plugin-vue";
import typescript from "rollup-plugin-typescript2";
import { rollup, OutputOptions } from "rollup";
import { compRoot } from "./utils/paths";
import { buildConfig } from "./utils/config";

function pathRewriter(format) {
  return (id: string) => {
    id = id.replaceAll("@z-plus/", `z-plus/${format}/`);
    return id;
  };
}
async function buildEachComponent() {
  const files = sync("*", {
    cwd: compRoot,
    onlyDirectories: true,
  });
  const builds = files.map(async (file: string) => {
    const entry = path.resolve(compRoot, file, "index.ts");
    const config = {
      input: entry,
      plugins: [nodeResolve(), vue(), typescript()],
      external: (id: string) => /^vue/.test(id) || /^@z-plus/.test(id),
    };
    const options = Object.values(buildConfig).map((config) => ({
      format: config.format,
      paths: pathRewriter(config.output.name),
      file: path.resolve(config.output.path, `components/${file}/index.js`),
    }));
    const bundle = await rollup(config);
    await Promise.all(
      options.map((option) => bundle.write(option as OutputOptions))
    );
  });

  return Promise.all(builds);
}
export const buildComponent = series(buildEachComponent);
```

> 给每个组件添加类型声明文件

```
pnpm i ts-morph -w -D
```

component-types.ts

```
import path from "path";
import fs from "fs/promises";
import * as vueCompiler from "@vue/compiler-sfc";
import { Project, SourceFile } from "ts-morph";
import glob from "fast-glob";
import { pathRewriter } from "./utils";
import { compRoot, outDir, projectRoot } from "./utils/paths";

async function genTypes() {
    const project = new Project({
        compilerOptions: {
            allowJs: true,
            declaration: true,
            emitDeclarationOnly: true,
            noEmitOnError: true,
            outDir: path.resolve(outDir, "types"),
            baseUrl: projectRoot,
            paths: {
                "@z-plus/*": ["packages/*"],
            },
            skipLibCheck: true,
            strict: false,
        },
        tsConfigFilePath: path.resolve(projectRoot, "tsconfig.json"),
        skipAddingFilesFromTsConfig: true,
    });

    const filePaths = await glob("**/*", {
        cwd: compRoot,
        onlyFiles: true,
        absolute: true,
    });

    const sourceFiles: SourceFile[] = [];
    await Promise.all(
        filePaths.map(async (file) => {
            if (file.endsWith(".vue")) {
                const content = await fs.readFile(file, "utf-8");
                const sfc = vueCompiler.parse(content);
                const { script } = sfc.descriptor;
                if (script) {
                    let content = script.content;
                    const sourceFile = project.createSourceFile(
                        file + ".ts",
                        content
                    );
                    sourceFiles.push(sourceFile);
                }
            } else if (file.endsWith(".ts")) {
                const sourceFile = project.addSourceFileAtPath(file);
                sourceFiles.push(sourceFile);
            }
        })
    );
    await project.emit({
```

```
        emitOnlyDtsFiles: true,
    });
    const tasks = sourceFiles.map(async (sourceFile: any) => {
        const emitOutput = sourceFile.getEmitOutput();
        const tasks = emitOutput.getOutputFiles().map(async (outputFile: any) =>
{
            const filepath = outputFile.getFilePath();
            await fs.mkdir(path.dirname(filepath), {
                recursive: true,
            });
            await fs.writeFile(filepath, pathRewriter("es")
(outputFile.getText()));
        });
        await Promise.all(tasks);
    });
    await Promise.all(tasks);
}
export { genTypes };
```

```
function copyTypes() {
  const src = path.resolve(outDir, "types/components/");
  const copy = (module) => {
    let output = path.resolve(
      outDir,
      buildConfig[module].output.name,
      "components/"
    );
    return () => run(`cp -r ${src}/* ${output}`);
  };
  return parallel(copy("esm"), copy("cjs"));
}

export const buildComponent = series(
    buildEachComponent,
    genTypes,
    copyTypes()
)
```

三.打包入口文件

```
async function buildComponentEntry() {
    const config = {
        input: path.resolve(compRoot, "index.ts"),
        plugins: [typescript()],
        external: () => true,
    };

    const bundle = await rollup(config);
    return Promise.all(
        Object.values(buildConfig)
            .map((config) => ({
                format: config.format,
                file: path.resolve(config.output.path, "components/index.js"),
            }))
            .map((config) => bundle.write(config as OutputOptions))
    );
```

```
}
export const buildComponent = series(
    buildEachComponent,
    buildComponentEntry,
    genTypes,
    copyTypes()
)
```

四.打包组件库入口`z-plus`

```
async function buildEntry() {
  const entryFiles = await fs.readdir(zpRoot, { withFileTypes: true });
  const entryPoints = entryFiles
    .filter((f) => f.isFile())
    .filter((f) => !["package.json"].includes(f.name))
    .map((f) => path.resolve(zpRoot, f.name));

  const config = {
    input: entryPoints,
    plugins: [nodeResolve(), vue(), typescript()],
    external: (id: string) => /^vue/.test(id) || /^@z-plus/.test(id),
  };
  const bundle = await rollup(config);
  return Promise.all(
    Object.values(buildConfig)
      .map((config) => ({
        format: config.format,
        dir: config.output.path,
        paths: pathRewriter(config.output.name),
      }))
      .map((option) => bundle.write(option as OutputOptions))
  );
}
export const buildFullComponent = parallel(buildFull, buildEntry);
```

> build/entry-types

```
export const genEntryTypes = async () => {
  const files = await glob("*.ts", {
    cwd: zpRoot,
    absolute: true,
    onlyFiles: true,
  });
  const project = new Project({
    compilerOptions: {
      declaration: true,
      module: ModuleKind.ESNext,
      allowJs: true,
      emitDeclarationOnly: true,
      noEmitOnError: false,
      outDir: path.resolve(outDir, "entry/types"),
      target: ScriptTarget.ESNext,
      rootDir: zpRoot,
      strict: false,
    },
    skipFileDependencyResolution: true,
```

```
    tsConfigFilePath: path.resolve(projectRoot, "tsconfig.json"),
    skipAddingFilesFromTsConfig: true,
  });
  const sourceFiles: SourceFile[] = [];
  files.map((f) => {
    const sourceFile = project.addSourceFileAtPath(f);
    sourceFiles.push(sourceFile);
  });
  await project.emit({
    emitOnlyDtsFiles: true,
  });
  const tasks = sourceFiles.map(async (sourceFile) => {
    const emitOutput = sourceFile.getEmitOutput();
    for (const outputFile of emitOutput.getOutputFiles()) {
      const filepath = outputFile.getFilePath();
      await fs.mkdir(path.dirname(filepath), { recursive: true });
      await fs.writeFile(
        filepath,
        outputFile.getText().replaceAll("@z-plus", "."),
        "utf8"
      );
    }
  });
  await Promise.all(tasks);
};
```

```
export const copyEntryTypes = () => {
    const src = path.resolve(outDir, "entry/types");
    const copy = (module) =>
        parallel(
            withTaskName(`copyEntryTypes:${module}`, () =>
                run(
                    `cp -r ${src}/* ${path.resolve(
                        outDir,
                        buildConfig[module].output.path
                    )}/`
                )
            )
        );
    return parallel(copy("esm"), copy("cjs"));
}
```

```
export default series(
    withTaskName('clean', () => run('rm -rf ./dist')),
    parallel(
        withTaskName("buildPackages", () =>
            run("pnpm run --filter ./packages --parallel build")
        ),
        withTaskName('buildFullComponent', () => run('pnpm run build
buildFullComponent')),
        withTaskName("buildComponent", () => run("pnpm run build
buildComponent"))
    ),
+   parallel(
+       copyEntryTypes()
+   )
)
```

五.拷贝`package.json`

```
import { series, parallel } from 'gulp'
import { copyEntryTypes } from './entry-types'
import { withTaskName, run } from './utils'
import { outDir, zpRoot } from './utils/paths'

const copySourceCode = () => async () => {
  await run(`cp ${zpRoot}/package.json ${outDir}/package.json`)
}

export * from './full-component'
export * from './component'
```